Northeastern University

DS 5220 – Lecture 5
KNN, Naive Bayes

Roi Yehoshua

Agenda

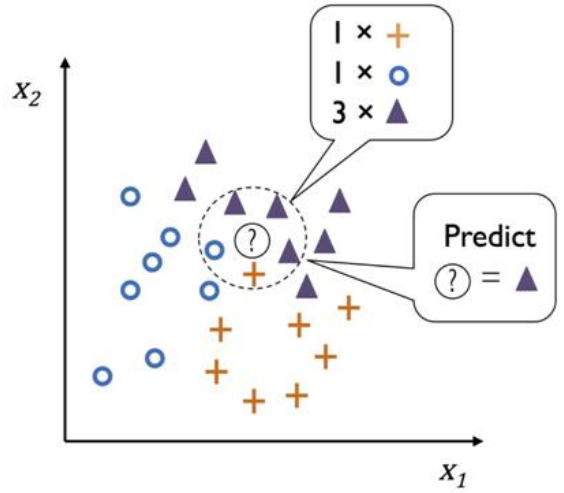
Northeastern University

- *k*-nearest neighbors
- Distance metrics
- Naive Bayes classifiers
- Laplace smoothing
- Text analysis and preprocessing
- Text classification

k-Nearest Neighbors (KNN)

Northeastern University

- KNN is a very simple, versatile and one of the topmost ML algorithms
- The label assigned to a sample is based on a majority vote of its *k* nearest neighbors
- KNN is a type of **instance-based learner** or **lazy learner**
  - Doesn't build an internal model from the data and defers all computation until prediction time
- KNN is also a **nonparametric method**
  - Doesn't assume anything about the form of the mapping from the features to the label



KNN Classification Algorithm

Northeastern University

Algorithm The k-nearest neighbors classification algorithm

Input: A set of training examples D, number of nearest neighbors k, and a test example $\mathbf{x}'$

for each training example $(\mathbf{x}, y) \in D$ **do**
  Compute $d(\mathbf{x}', \mathbf{x})$, the distance between $\mathbf{x}'$ and $\mathbf{x}$
Select $D_z \subseteq D$, the set of k closest training examples to $\mathbf{x}'$
$y' = \text{argmax}_v \sum_{(\mathbf{x}_i, y_i) \in D_z} \mathbb{1}\{y_i = v\}$
return y'

- 1{·} is an indicator function that returns 1 if its argument is true, and 0 otherwise
  - e.g., 1{2 = 3} = 0 and 1{3 = 5 - 2} = 1

Example

Northeastern University

- Given the following data set:

i	x_1	x_2	x_3	y
1	1	4	1	1
2	1	0	-2	0
3	0	0	1	0
4	-1	4	0	1
5	-1	-1	1	1
6	1	2	3	1
7	0	-4	0	0
8	1	0	-3	0

- Classify the vector (1, 0, 1) using KNN with $k = 3$ and using Euclidean distance

Example

Northeastern University

- The distances of (1,0,1) from the training samples are:

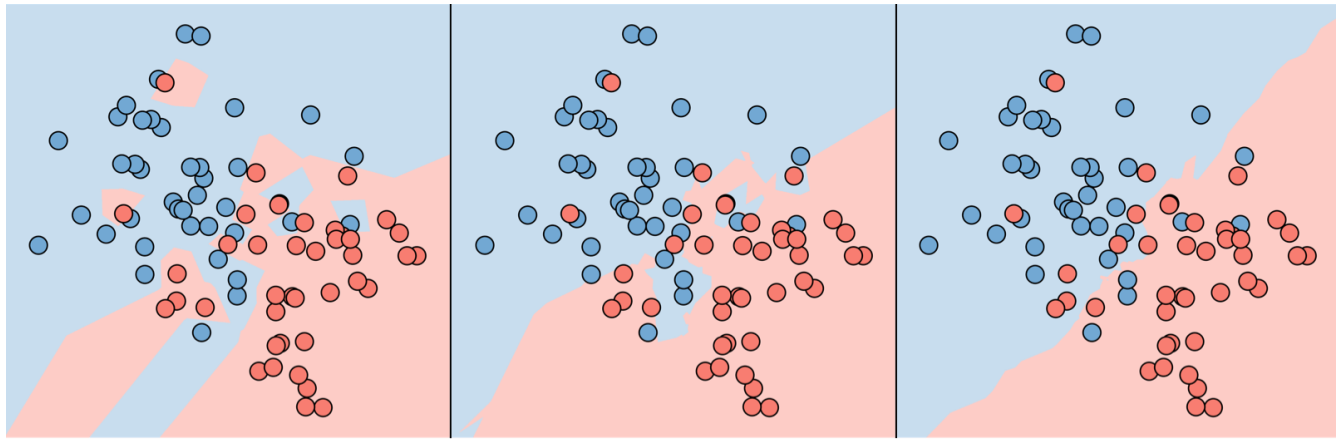
i	x_1	x_2	x_3	y	distance from (1, 0, 1)
1	1	4	1	1	$(1-1)^2 + (4-0)^2 + (1-1)^2 = 16$
2	1	0	-2	0	$(1-1)^2 + (0-0)^2 + (-2-1)^2 = 9$
3	0	0	1	0	$(0-1)^2 + (0-0)^2 + (1-1)^2 = 1$
4	-1	4	0	1	$(-1-1)^2 + (4-0)^2 + (0-1)^2 = 19$
5	-1	-1	1	1	$(-1-1)^2 + (-1-0)^2 + (1-1)^2 = 5$
6	1	2	3	1	$(1-1)^2 + (2-0)^2 + (3-1)^2 = 8$
7	0	-4	0	0	$(0-1)^2 + (-4-0)^2 + (0-1)^2 = 18$
8	1	0	-3	0	$(1-1)^2 + (0-0)^2 + (-3-1)^2 = 16$

- The majority label of the 3 closest neighbors is 1, therefore the predicted label is 1

Choosing the Number of Neighbors *k*

Northeastern University

- The choice of *k* has a large impact on the performance of KNN
- If *k* is too small, the result can be sensitive to noise (mislabeled training examples)
- If *k* is too large, the neighborhood may include too many points from other classes
- Rule of thumb: choose $k = \sqrt{n}$, where *n* is the number of points in the training set
  - Usually works when the number of samples is high enough



Distance Measures

Northeastern University

- The choice of the distance measure is another important consideration
- Depends on the type of the features (numerical, categorical, text, …)
- Features have to be scaled to prevent distance measures from being dominated by one of the features

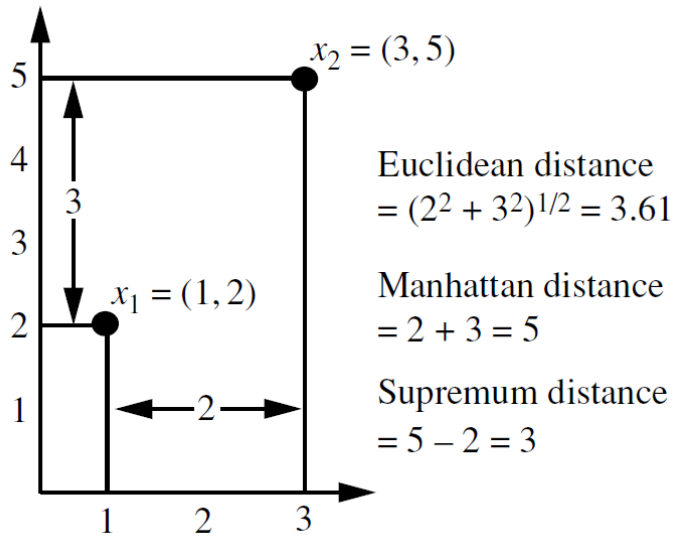
Minkowski Distance

Northeastern University

- Metric used for real-valued vectors
- Minkowski distance is a generalization of Euclidean distance

$$d(\mathbf{x}, \mathbf{y}) = \left(\sum_{j=1}^{d} |x_j - y_j|^p \right)^{1/p}$$

- Common examples:
  - p = 1: Manhattan (city block, taxicab) distance
  - p = 2: Euclidean distance
  - $p \to \infty$: Supremum distance
    - The maximum difference between components of the vectors



Jaccard Similarity

Northeastern University

- Metric used for **asymmetric binary vectors**
  - where the agreement of two 1s is considered more significant than of two 0s
  - e.g., when comparing users in a recommendation system where each user is represented by a binary vector of the items that they purchased
- Jaccard similarity is defined as

$$J(\mathbf{x}, \mathbf{y}) = \frac{\text{number of 11 matches}}{\text{number of non-both-zero attributes}} = \frac{M_{11}}{M_{01} + M_{10} + M_{11}}$$

  - M_{11} = the number of attributes where **x** is 1 and **y** is 1
  - M_{01} = the number of attributes where **x** is 0 and **y** is 1
  - M_{10} = the number of attributes where **x** is 1 and **y** is 0
- Jaccard distance is defined as

$$d_J(\mathbf{x}, \mathbf{y}) = 1 - J(\mathbf{x}, \mathbf{y})$$

Jaccard Similarity: Example

Northeastern University

- Given the vectors:

$$\mathbf{x} = (1, 0, 0, 1, 0, 0, 0, 1, 0, 0)^T$$
$$\mathbf{y} = (0, 0, 0, 1, 0, 0, 1, 1, 0, 1)^T$$

- The Jaccard similarity between them is:

$$J(\mathbf{x}, \mathbf{y}) = \frac{M_{11}}{M_{01} + M_{10} + M_{11}} = \frac{2}{2 + 1 + 2} = \frac{2}{5} = 0.4$$

- And the Jaccard distance is:

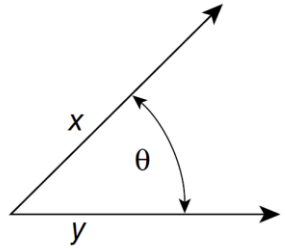
$$d_J(\mathbf{x}, \mathbf{y}) = 1 - 0.4 = 0.6$$

Cosine Similarity

Northeastern University

- Documents are often represented as word count vectors, where each feature represents the frequency of a word in the document
- This is a very sparse representation, since it has very few non-zero attributes
- We need a similarity measure that ignores 0-0 matches (like Jaccard similarity) but handles non-binary vectors
- Cosine similarity is the cosine of the angle between the two document vectors:

$$\cos(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{||\mathbf{x}||||\mathbf{y}||}$$



  - The smaller the angle, the higher the similarity
- Cosine similarity doesn't depend on the magnitude of the two data objects
  - which is desirable for comparing documents of different lengths

Cosine Similarity: Example

Northeastern University

- Given the vectors:

$$\mathbf{x} = (3, 2, 0, 5, 0, 0, 0, 2, 0, 0)$$
$$\mathbf{y} = (1, 0, 0, 0, 0, 0, 0, 1, 0, 2)$$

- The cosine similarity between them is:

$$\mathbf{x} \cdot \mathbf{y} = 3 \cdot 1 + 2 \cdot 0 + 0 \cdot 0 + 5 \cdot 0 + 0 \cdot 0 + 0 \cdot 0 + 0 \cdot 0 + 2 \cdot 1 + 0 \cdot 0 + 0 \cdot 2 = 5$$
$$||\mathbf{x}|| = \sqrt{3 \cdot 3 + 2 \cdot 2 + 0 \cdot 0 + 5 \cdot 5 + 0 \cdot 0 + 0 \cdot 0 + 0 \cdot 0 + 2 \cdot 2 + 0 \cdot 0 + 0 \cdot 0} = \sqrt{42} = 6.481$$
$$||\mathbf{y}|| = \sqrt{1 \cdot 1 + 0 \cdot 0 + 0 \cdot 0 + 0 \cdot 0 + 0 \cdot 0 + 0 \cdot 0 + 0 \cdot 0 + 1 \cdot 1 + 0 \cdot 0 + 2 \cdot 2} = \sqrt{6} = 2.449$$

$$\cos(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{||\mathbf{x}||||\mathbf{y}||} = \frac{5}{6.481 \cdot 2.449} = 0.315$$

Distance-Weighted Voting

Northeastern University

- Often the closer neighbors more reliably indicate the class of the instance
- Thus, we can weight each neighbor's vote, so that nearer neighbors contribute more to the average than the more distant ones
- In this case, the class label is determined as follows:

$$y' = \underset{v}{\operatorname{argmax}} \sum_{(\mathbf{x}_i, y_i) \in D_z} w_i \cdot \mathbb{1}\{y_i = v\}$$

- A common weighting scheme is to give each neighbor a weight of 1/*d*, where *d* is its distance from the test point $w_i = \frac{1}{d(\mathbf{x}', \mathbf{x}_i)}$
- This method is less sensitive to the choice of *k*

Example

Northeastern University

- The distances of (1,0,1) from the training samples are:

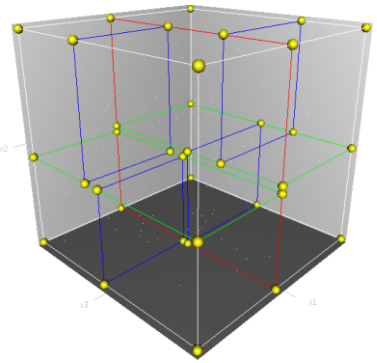
i	x_1	x_2	x_3	y	distance from (1, 0, 1)	weight
1	1	4	1	1	$(1-1)^2 + (4-0)^2 + (1-1)^2 = 16$	
2	1	0	-2	0	$(1-1)^2 + (0-0)^2 + (-2-1)^2 = 9$	
3	0	0	1	0	$(0-1)^2 + (0-0)^2 + (1-1)^2 = 1$	1/1 = 1
4	-1	4	0	1	$(-1-1)^2 + (4-0)^2 + (0-1)^2 = 19$	
5	-1	-1	1	1	$(-1-1)^2 + (-1-0)^2 + (1-1)^2 = 5$	1/5 = 0.2
6	1	2	3	1	$(1-1)^2 + (2-0)^2 + (3-1)^2 = 8$	1/8 = 0.125
7	0	-4	0	0	$(0-1)^2 + (-4-0)^2 + (0-1)^2 = 18$	
8	1	0	-3	0	$(1-1)^2 + (0-0)^2 + (-3-1)^2 = 16$	

- The majority label of the 3 closest neighbors is 1, therefore the predicted label is 1

KD-Tree

Northeastern University

- Allows a more efficient retrieval of the closest neighbors
- A KD-tree is a binary tree structure that recursively splits the space into half-spaces
- Every node in the tree represents a *d*-dimensional point in space
- Every non-leaf node is associated with one of the *d* dimensions
  - Points that have a smaller value in that dimension will be in the left subtree of that node
  - Points with a larger value will be in its right subtree
- Points that are close to each other are usually stored at nearby nodes
- For small *d* ($d \leq 20$), the query time of KD-tree is approximately $O(d \log n)$
- For larger *d*, the cost increases nearly to $O(dn)$



The Curse of Dimensionality

Northeastern University

- KNN may not work as well when there is a large number of features (dimensions)
- When dimensionality increases, data becomes increasingly sparse in space
  - Even the closest neighbors can be too far away to give a good estimate
- Solution: use dimensionality reduction

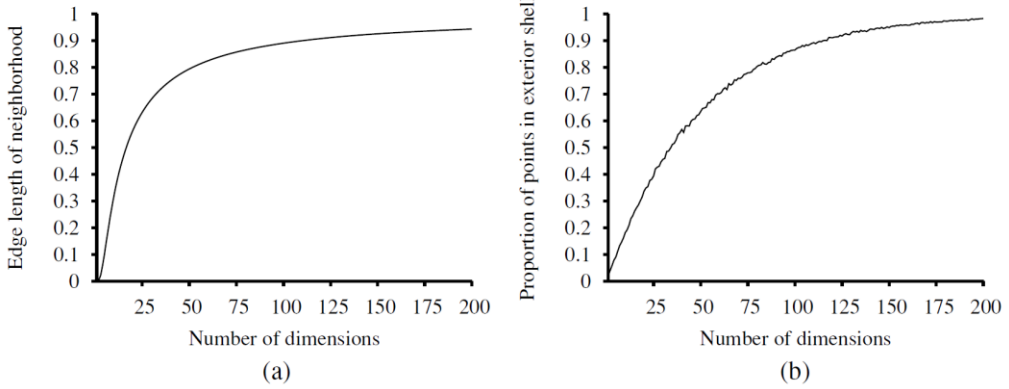


Figure 18.27 The curse of dimensionality: (a) The length of the average neighborhood for 10-nearest-neighbors in a unit hypercube with 1,000,000 points, as a function of the number of dimensions. (b) The proportion of points that fall within a thin shell consisting of the outer 1% of the hypercube, as a function of the number of dimensions. Sampled from 10,000 randomly distributed points.

KNN in Scikit-Learn

Northeastern University

class `sklearn.neighbors.` **`KNeighborsClassifier`**(*n_neighbors=5, *, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None, **kwargs*) [source]

Argument	Description
n_neighbors	Number of neighbors to use (default = 5)
weights	weight function used in prediction. Possible values: 'uniform': uniform weights. All points in each neighborhood are weighted equally (default). 'distance': weight points by the inverse of their distance. [callable]: a user-defined function
algorithm	Algorithm used to compute the nearest neighbors: 'ball_tree', 'kd_tree', 'brute', or 'auto'. 'auto' will choose the most appropriate algorithm based on the data set.
leaf_size	Leaf size passed to BallTree or KDTree (default = 30)
p	Power parameter for the Minkowski metric.
metric	The distance metric to use. A list of available metrics can be found here.

KNN in Scikit-Learn

Northeastern University

- Let's train KNN classifiers with $k = 5$ and $k = 20$ on the Iris data set using only the first two features (sepal length and sepal width)

```
from sklearn.datasets import load_iris

iris = load_iris()
X = iris.data[:, :2]  # We only take the first two features
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y)
```

```
from sklearn.neighbors import KNeighborsClassifier

clf = KNeighborsClassifier(n_neighbors=5)
clf.fit(X_train, y_train)

clf2 = KNeighborsClassifier(n_neighbors=20)
clf2.fit(X_train, y_train)
```

```
KNeighborsClassifier(n_neighbors=20)
```

KNN in Scikit-Learn

Northeastern University

- The accuracy of the classifiers on the training and test sets:

```
print(f'Training set accuracy: {clf.score(X_train, y_train):.4f}')
print(f'Test set accuracy: {clf.score(X_test, y_test):.4f}')
```

```
Training set accuracy: 0.8393
Test set accuracy: 0.7632
```

```
print(f'Training set accuracy: {clf2.score(X_train, y_train):.4f}')
print(f'Test set accuracy: {clf2.score(X_test, y_test):.4f}')
```

```
Training set accuracy: 0.8214
Test set accuracy: 0.8158
```

KNN in Scikit-Learn

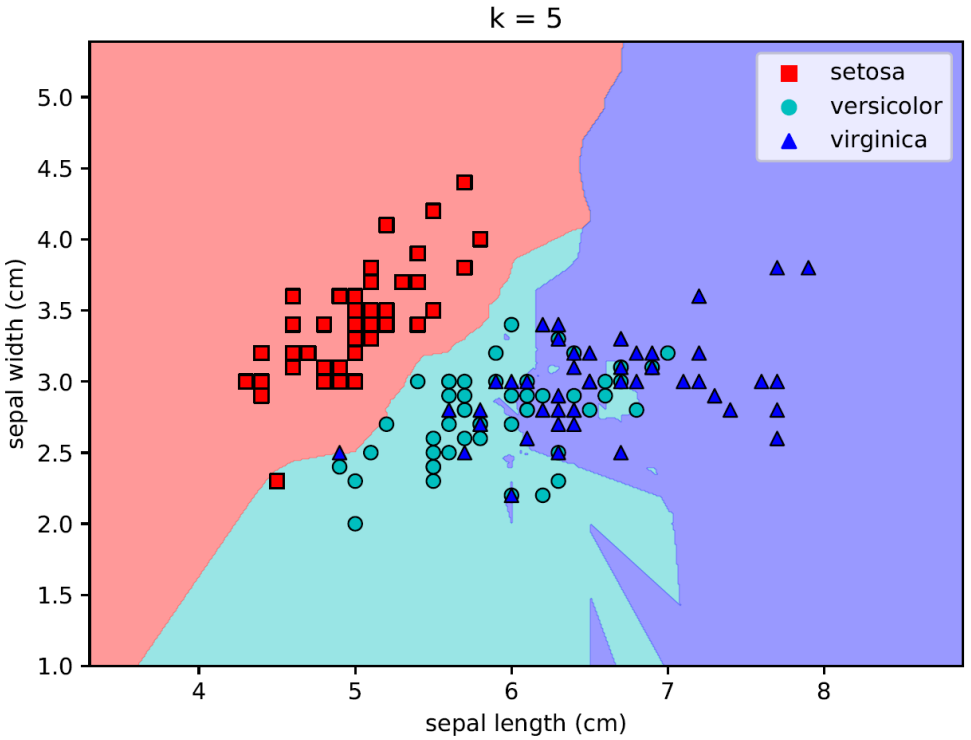
Northeastern University

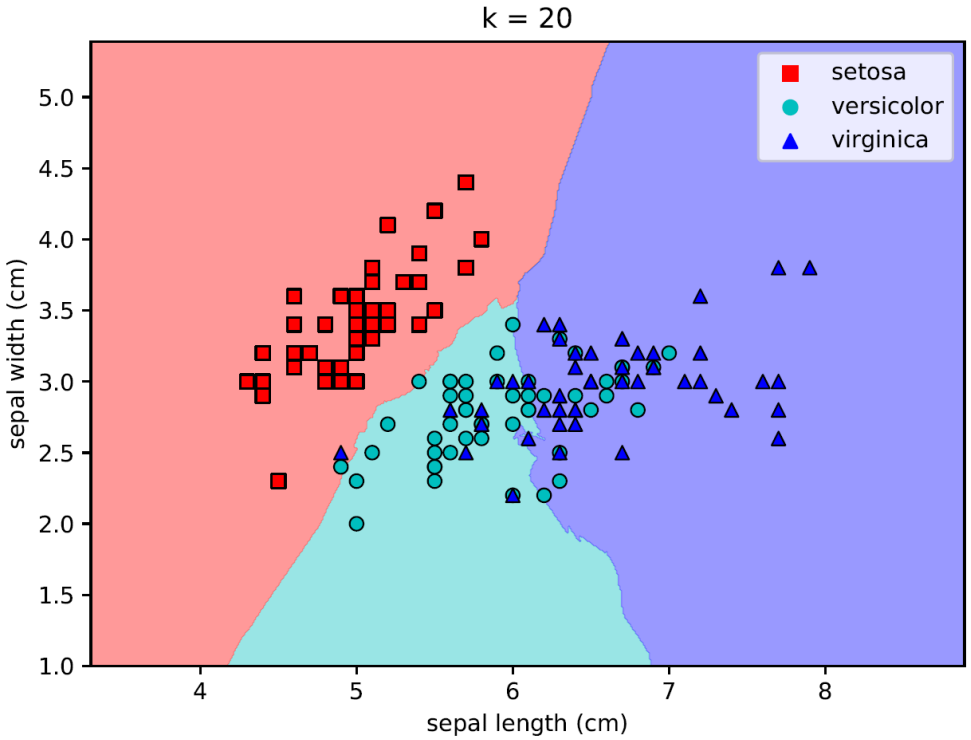
- Let's plot the decision boundaries of the classifiers:

```
from matplotlib.colors import ListedColormap

def plot_decision_boundaries(clf, X, y, feature_names, class_names, h=0.01):
    colors = ['r', 'c', 'b']
    cmap = ListedColormap(colors)

    # Assign a color to each point in the mesh [x_min, x_max]x[y_min, y_max]
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, alpha=0.4, cmap=cmap)

    # Plot also the sample points
    sns.scatterplot(X[:, 0], X[:, 1], hue=class_names[y], style=class_names[y],
                    palette=colors, markers=('s', 'o', '^'), edgecolor='black')
    plt.xlabel(feature_names[0])
    plt.ylabel(feature_names[1])
    plt.legend()
```

KNN in Scikit-Learn

- Let's plot the decision boundaries of the classifiers:



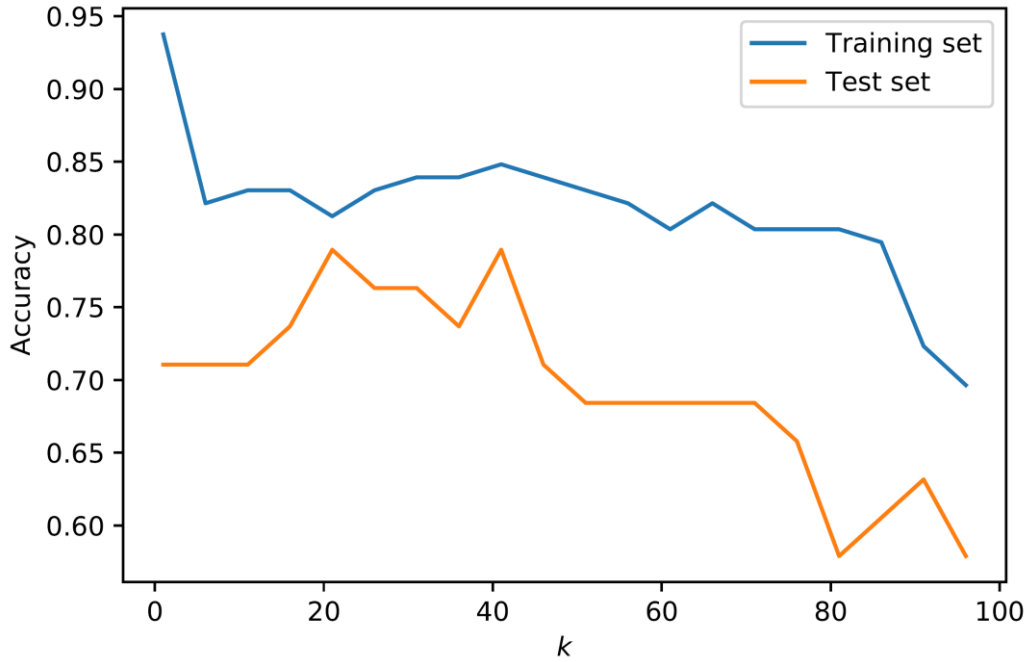


KNN in Scikit-Learn

Northeastern University

- Let's examine the effect of changing the number of neighbors on the accuracy:

```
n_neighbors, train_scores, test_scores = [], [], []

for n in range(1, 100, 5):
    n_neighbors.append(n)
    clf = KNeighborsClassifier(n_neighbors=n)
    clf.fit(X_train, y_train)
    train_scores.append(clf.score(X_train, y_train))
    test_scores.append(clf.score(X_test, y_test))

plt.plot(n_neighbors, train_scores, label='Training set')
plt.plot(n_neighbors, test_scores, label='Test set')
plt.xlabel('$k$')
plt.ylabel('Accuracy')
plt.legend()
plt.savefig('figures/knn_results.pdf')
```



Nearest Neighbors Regression

Northeastern University

- The KNN algorithm can also be used for regression
- The label assigned to a new data point is computed based on the mean of the labels of its nearest neighbors
- This algorithm is implemented in the class KNeighborsRegressor

class `sklearn.neighbors.` **`KNeighborsRegressor`**(*n_neighbors=5, *, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None, **kwargs*) [source]

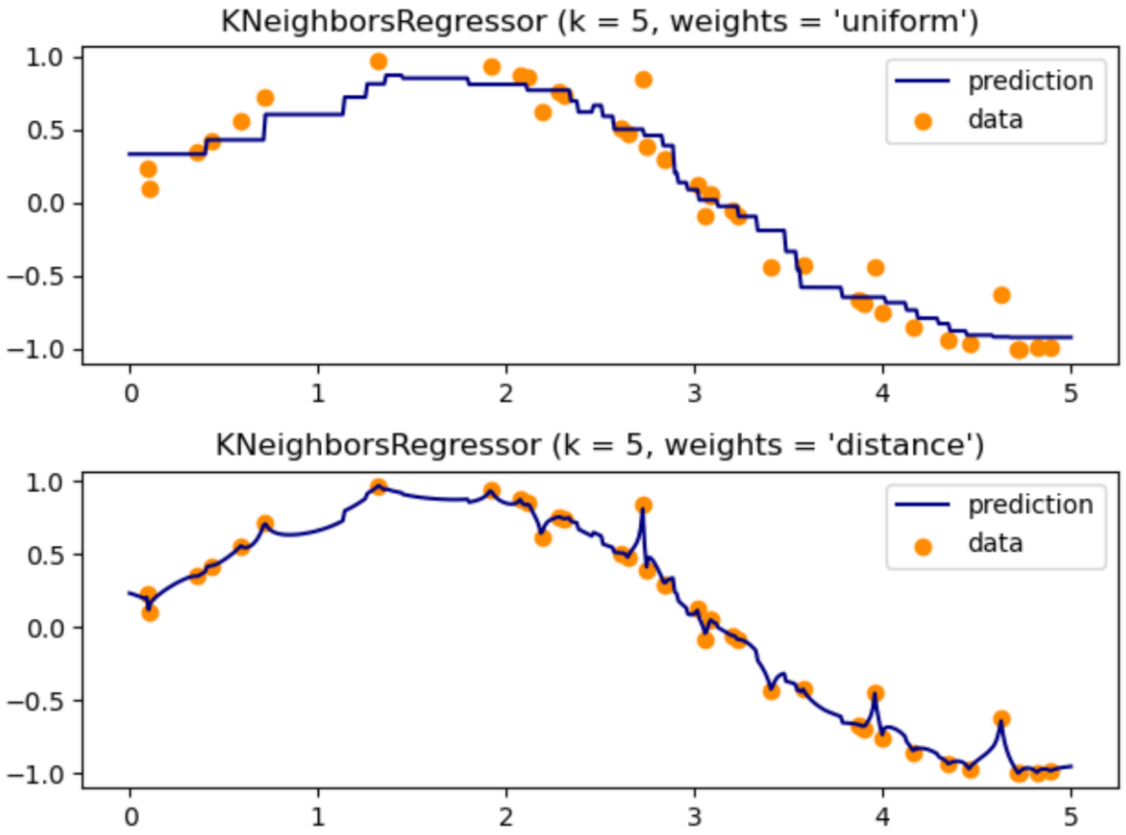
```
X = np.arange(5).reshape(-1, 1)
y = [2, 7, 4, 6, 8]
```

```
from sklearn.neighbors import KNeighborsRegressor

reg = KNeighborsRegressor(n_neighbors=2)
reg.fit(X, y)
reg.predict([[1.5]])
```

```
array([5.5])
```

Nearest Neighbors Regression

- Regression of the function $y = \sin(x)$ (+noise) using 40 random points in [0, 5]:



KNN Summary

Northeastern University

Pros

- Doesn't require to build a model
  - No information loss
  - Training is very fast
- Can produce decision boundaries of arbitrary shape
- Easy to implement
- Performs well in many problems
- Immediately adapts to new data
- Can be used for both classification and regression
- Small number of hyperparameters

Cons

- Slow at prediction time
- Need to store all the training examples
- Susceptible to noise/outliers (for small *k*)
- Sensitive to *k* and the distance measure
- Curse of dimensionality
- Doesn't perform well on imbalanced data sets
- Limited interpretability
- Has difficulty in handling missing values
- Requires scaling of the data

The Naive Bayes Model

Northeastern University

- Given a sample (**x**, *y*), compute the **class posterior probability** using Bayes' rule:

$$P(y = k|\mathbf{x}) = \frac{P(\mathbf{x}|y = k)P(y = k)}{P(\mathbf{x})}$$

- The **class prior probability** can be estimated from the frequency of class *k* in the data

$$P(y = k) = \frac{n_k}{n}$$

n_k: number of samples of class *k* in the training set

n: total number of training samples

- *P*(**x**) is a normalizing factor: $P(\mathbf{x}) = \sum_{k=1}^{K} P(\mathbf{x}|y = k)P(y = k)$

- How to estimate *P*(**x** | *y* = *k*) from the data?
  - Problem: Need to estimate an exponential number of probabilities!
  - e.g., if there are *m* binary features, we need to estimate $2^m K$ probabilities

Naive Bayes Classification

Northeastern University

- **Naive Bayes assumption**: the features are conditionally independent given the class

$$P(\mathbf{x}|y=k) = P(x_1|y=k) \cdot P(x_2|y=k) \cdots P(x_m|y=k) = \prod_{j=1}^{m} P(x_j|y=k)$$

- Therefore we can write the posterior probabilities of the classes as:

$$P(y=k|\mathbf{x}) = \frac{P(y=k)\prod_{j=1}^{m} P(x_j|y=k)}{P(\mathbf{x})}$$

  - Now we only need to estimate *m* parameters instead of 2^m for each class

- If we only care about the labels (and not the probabilities):

$$\hat{y} = \underset{k}{\operatorname{argmax}}\, P(y=k)\prod_{j=1}^{m} P(x_j|y=k)$$

- The different NB classifiers differ by the assumption they make on $P(x_j|y)$

Bernoulli Naive Bayes

Northeastern University

- Assumption: each feature is distributed according to a Bernoulli distribution
- Example: in text classification, each feature x_j may represent the occurrence or absence of the *j*th word from the vocabulary in the text
- The parameters of the model are estimated using relative frequency counting
  - For each feature *j* and class *k*:

$$P(x_j = 1|y = k) = \frac{n_{jk}}{n_k}$$

number of times feature *j* appears in samples of class *k*

number of samples in class *k*

Categorical Naive Bayes

Northeastern University

- Assumption: each feature has a categorical distribution
  - Generalization of Bernoulli distribution to *m* possible categories
- The parameters of the model are:
  - For each feature *j*, class *k* and category *v*

$$P(x_j = v|y = k) = \frac{n_{jvk}}{n_k}$$

n_{jvk}: number of times feature *j* has the value *v* in samples of class *k*

n_k: number of samples in class *k*

Categorical Naive Bayes Example

Northeastern University

- We'd like to decide whether to go out play tennis based on weather conditions

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Given a new sample:
x = (Outlook = sunny, Temperature = cool, Humidity = High, Wind = strong)
Predict whether to play tennis or not.

Categorical Naive Bayes Example

Northeastern University

- We first estimate the class prior probabilities based on their frequency in the data:

$$P(\text{PlayTennis} = \text{Yes}) = 9/14 = 0.643$$
$$P(\text{PlayTennis} = \text{No}) = 5/14 = 0.357$$

- We now estimate the conditional probabilities of the features in the new sample:

$$P(\text{Outlook} = \text{Sunny}|\text{PlayTennis} = \text{Yes}) = 2/9 = 0.222$$
$$P(\text{Outlook} = \text{Sunny}|\text{PlayTennis} = \text{No}) = 3/5 = 0.6$$
$$P(\text{Temperature} = \text{Cool}|\text{PlayTennis} = \text{Yes}) = 3/9 = 0.333$$
$$P(\text{Temperature} = \text{Cool}|\text{PlayTennis} = \text{No}) = 1/5 = 0.2$$
$$P(\text{Humidity} = \text{High}|\text{PlayTennis} = \text{Yes}) = 3/9 = 0.333$$
$$P(\text{Humidity} = \text{High}|\text{PlayTennis} = \text{No}) = 4/5 = 0.8$$
$$P(\text{Wind} = \text{Strong}|\text{PlayTennis} = \text{Yes}) = 3/9 = 0.333$$
$$P(\text{Wind} = \text{Strong}|\text{PlayTennis} = \text{No}) = 3/5 = 0.6$$

Categorical Naive Bayes Example

Northeastern University

- Therefore, the class posterior probabilities are:

$$P(\text{Yes}|\mathbf{x}) = \alpha P(\text{Yes})P(\text{O} = \text{Sunny}|\text{Yes})P(\text{T} = \text{Cool}|\text{Yes})P(\text{H} = \text{High}|\text{Yes})P(\text{W} = \text{Strong}|\text{Yes})$$
$$= \alpha 0.643 \cdot 0.222 \cdot 0.333 \cdot 0.333 \cdot 0.333 = 0.0053\alpha$$

$$P(\text{No}|\mathbf{x}) = \alpha P(\text{No})P(\text{O} = \text{Sunny}|\text{No})P(\text{T} = \text{Cool}|\text{No})P(\text{H} = \text{High}|\text{No})P(\text{W} = \text{Strong}|\text{No})$$
$$= \alpha 0.357 \cdot 0.6 \cdot 0.2 \cdot 0.8 \cdot 0.6 = 0.0206\alpha$$

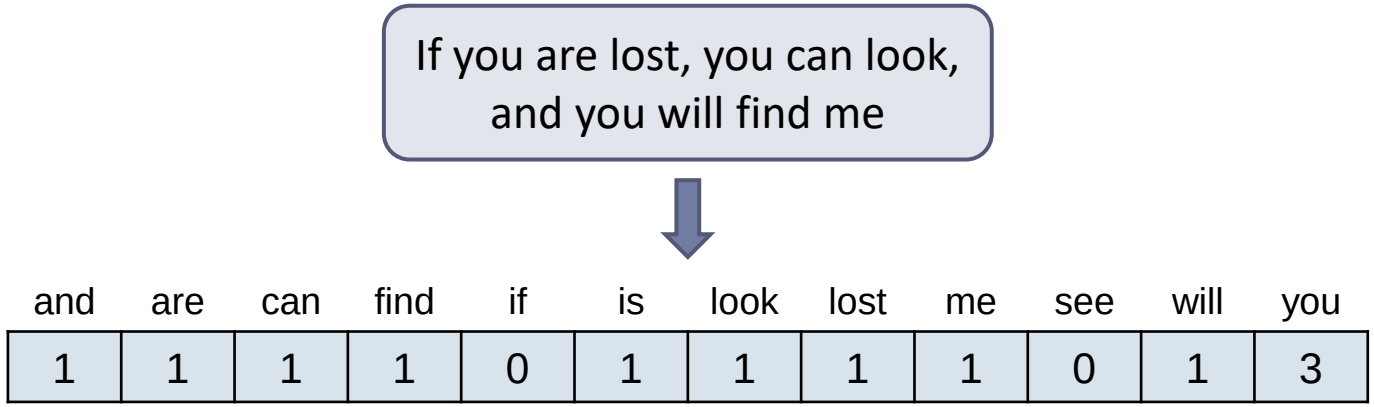
  - $\alpha = 1/P(\mathbf{x})$ is a constant term

- Since $P(\text{No}|\mathbf{x}) > P(\text{Yes}|\mathbf{x})$, the sample is classified as PlayTennis = No

Multinomial Naive Bayes

Northeastern University

- Assumption:
  - There is only one categorical feature x that can take one of m categories
  - Each feature vector $(x_1, \ldots, x_m)$ is a histogram, where x_j counts the number of times x had the value j in that particular instance
- The multinomial distribution is a generalization of the binomial distribution to more than two possible outcomes in each trial
- Example: a bag-of-words model in text classification



Multinomial Naïve Bayes

Northeastern University

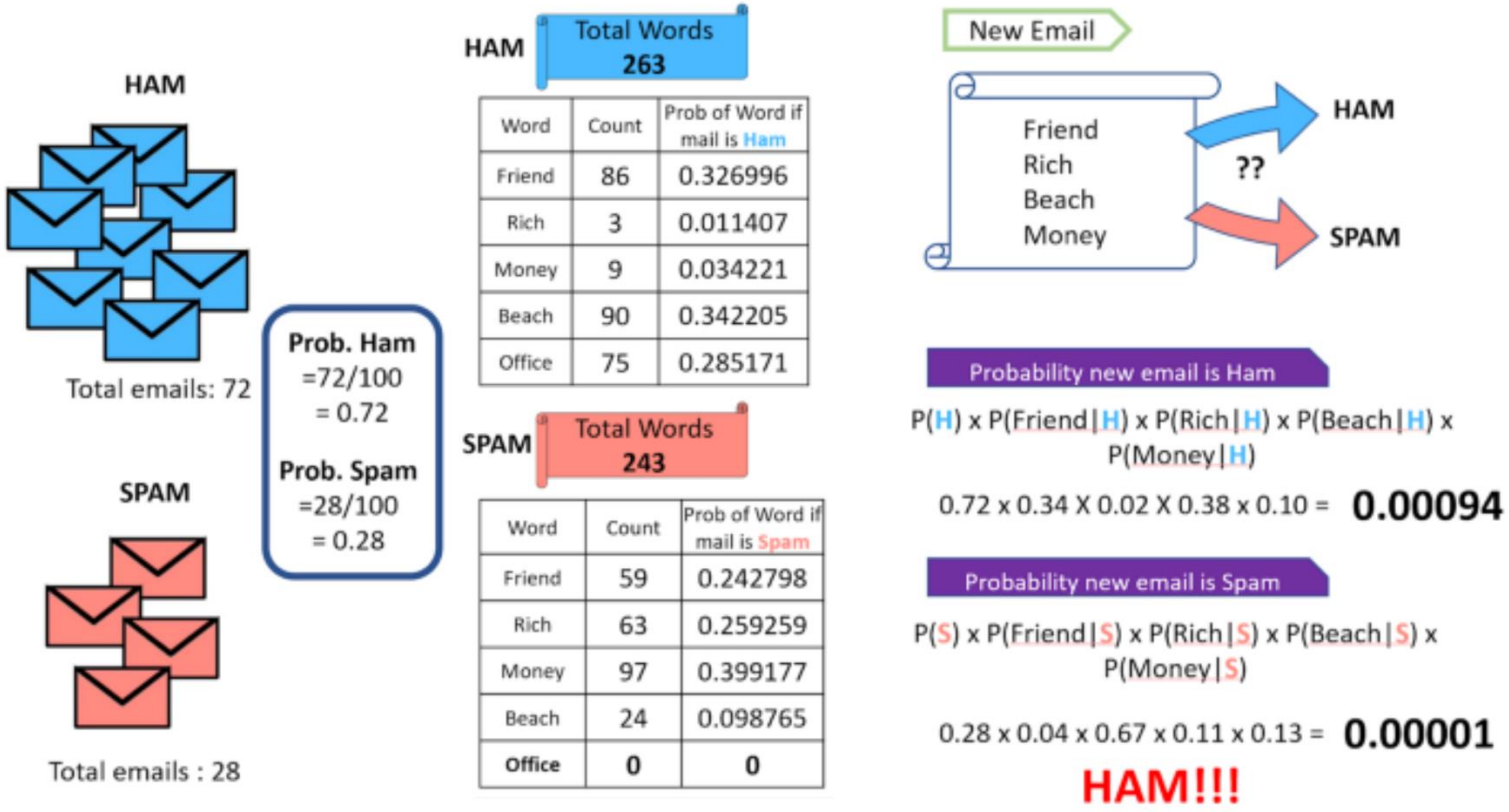
- The parameters of the model are:
  - For each class *k* and category *v*

$$P(x = v|y = k) = \frac{n_{vk}}{n_k}$$

n_{vk}: number of times *x* got the value *v* (e.g., the word v appeared in the text) in samples of class *k*

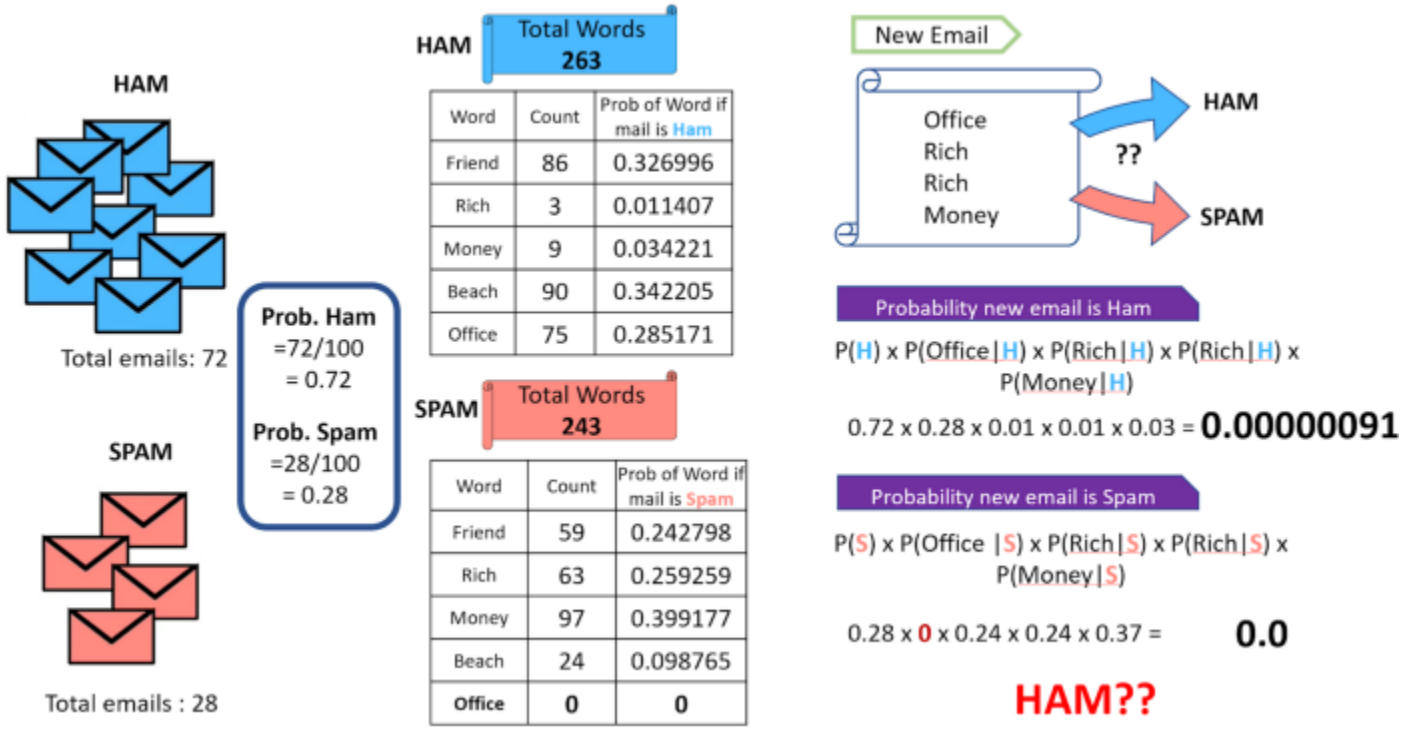
n_k: total number of samples (e.g., words) in class *k*

Example: Spam Filter



Source: https://www.atoti.io/how-to-solve-the-zero-frequency-problem-in-naive-bayes/

The Zero Frequency Problem



Laplace Smoothing

Northeastern University

- Add a small sample correction in all the probability estimates, such that no probability is never set to exactly zero
- In categorical Naive Bayes:

$$P(x_j = v|y = k) = \frac{n_{jvk} + \alpha}{n_k + \alpha n_j}$$

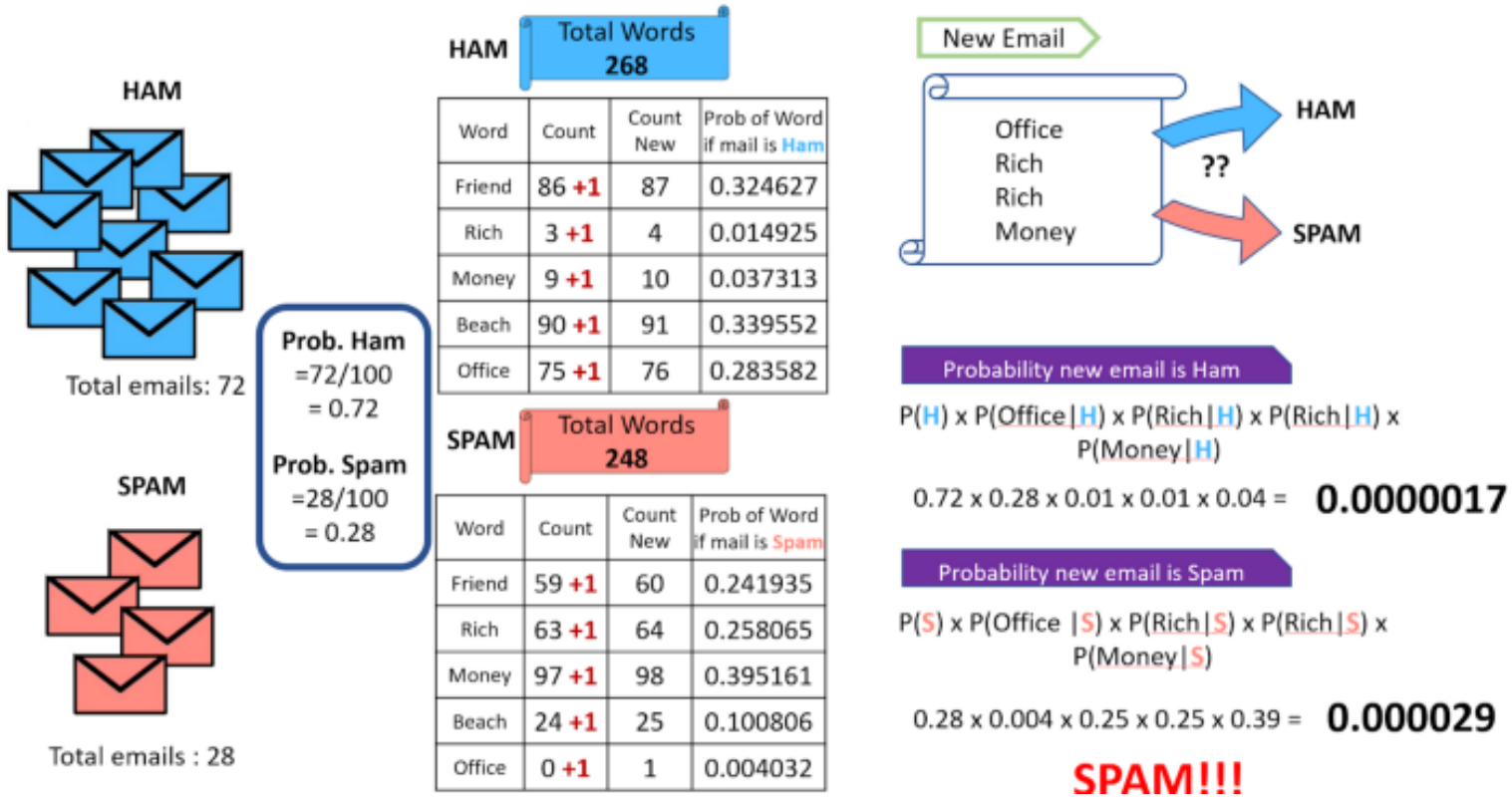
- In multinomial Naive Bayes:

$$P(x = v|y = k) = \frac{n_{vk} + \alpha}{n_k + \alpha n}$$

- α is a smoothing parameter
  - Setting $\alpha = 1$ is called **Laplace smoothing** (the most common one
  - $\alpha < 1$ is called Lidstone smoothing

Example: Laplace Smoothing

- Solving the zero frequency problem



Word	Count	Count New	Prob of Word if mail is Ham
Friend	86 +1	87	0.324627
Rich	3 +1	4	0.014925
Money	9 +1	10	0.037313
Beach	90 +1	91	0.339552
Office	75 +1	76	0.283582

Word	Count	Count New	Prob of Word if mail is Spam
Friend	59 +1	60	0.241935
Rich	63 +1	64	0.258065
Money	97 +1	98	0.395161
Beach	24 +1	25	0.100806
Office	0 +1	1	0.004032

Naïve Bayes in Scikit-Learn

Northeastern University

- The module sklearn.naive_bayes contains implementation for all three models:

class `sklearn.naive_bayes.` **`BernoulliNB`**(*, *alpha=1.0*, *binarize=0.0*, *fit_prior=True*, *class_prior=None*) [source]

class `sklearn.naive_bayes.` **`CategoricalNB`**(*, *alpha=1.0*, *fit_prior=True*, *class_prior=None*, *min_categories=None*) [source]

class `sklearn.naive_bayes.` **`MultinomialNB`**(*, *alpha=1.0*, *fit_prior=True*, *class_prior=None*) [source]

- The argument alpha specifies the Laplace smoothing parameter
  - Use 0 for no smoothing

Example: Text Classification

Northeastern University

- In the following example we'll use a Naïve Bayes classifier to classify text documents into different categories
- The "Twenty Newsgroups" dataset contains around 20,000 newsgroups posts, partitioned (nearly) evenly across 20 topics
- It has become popular for experiments in text applications of ML techniques, such as text classification and text clustering
- You can download the data set using the function fetch_20newsgroups():

```
from sklearn.datasets import fetch_20newsgroups

twenty_train = fetch_20newsgroups(subset='train')
twenty_test = fetch_20newsgroups(subset='test')
```

  - The training and test sets need to be downloaded separately
  - It may take a few minutes to download the data the first time you try to load it:

Example: Text Classification

Northeastern University

- Let's take a look at the list of category names:

```
twenty_train.target_names
```

```
['alt.atheism',
 'comp.graphics',
 'comp.os.ms-windows.misc',
 'comp.sys.ibm.pc.hardware',
 'comp.sys.mac.hardware',
 'comp.windows.x',
 'misc.forsale',
 'rec.autos',
 'rec.motorcycles',
 'rec.sport.baseball',
 'rec.sport.hockey',
 'sci.crypt',
 'sci.electronics',
 'sci.med',
 'sci.space',
 'soc.religion.christian',
 'talk.politics.guns',
 'talk.politics.mideast',
 'talk.politics.misc',
 'talk.religion.misc']
```

Example: Text Classification

Northeastern University

- The text files themselves are loaded in the **data** attribute
- The category index of each document is stored in the **target** attribute

```
print(twenty_train.data[0])
```

```
From: lerxst@wam.umd.edu (where's my thing)
Subject: WHAT car is this!?
Nntp-Posting-Host: rac3.wam.umd.edu
Organization: University of Maryland, College Park
Lines: 15

 I was wondering if anyone out there could enlighten me on this car I saw
the other day. It was a 2-door sports car, looked to be from the late 60s/
early 70s. It was called a Bricklin. The doors were really small. In additi
on,
the front bumper was separate from the rest of the body. This is
all I know. If anyone can tellme a model name, engine specs, years
of production, where this car is made, history, or whatever info you
have on this funky looking car, please e-mail.

Thanks,
- IL
   ---- brought to you by your neighborhood Lerxst ----
```

```
twenty_train.target_names[twenty_train.target[0]]
```

```
'rec.autos'
```

Text Analysis

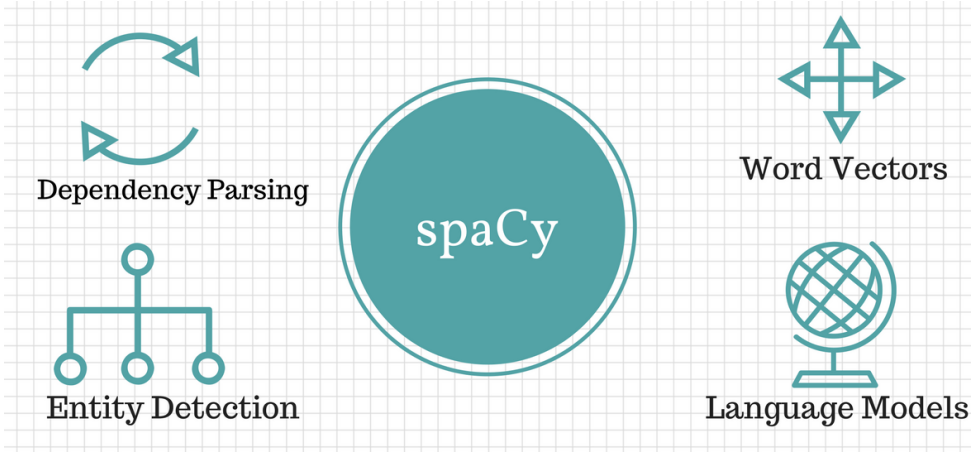
Northeastern University

- In order to perform ML on text documents, we first need to turn the text content into numerical feature vectors (or “vectorize” the text)
- This process typically involves two steps:
  - Pre-process, clean, and normalize the text
  - Transform the text documents into numerical representations

Python Libraries for Text Analysis

Northeastern University

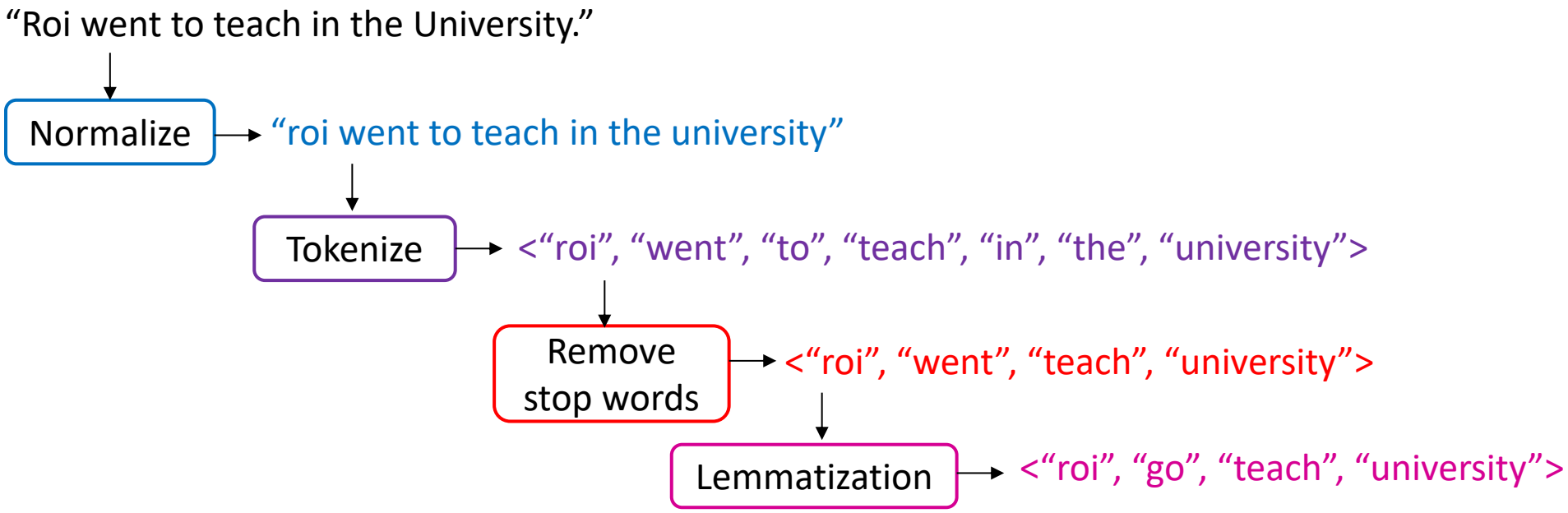
- Many of the basic text analysis operations can be done in Scikit-Learn (e.g., tokenization, removing stop words, TF-IDF vectorization)
- For more advanced operations (e.g., lemmatization, parsing, word embeddings) you can use the following Python libraries:
  - NLTK (Natural Language Toolkit) https://www.nltk.org/
  - spaCy https://spacy.io/
- Both can be installed via pip



Text Preprocessing

Northeastern University

- Common pre-processing techniques include:
  - Normalization (e.g., lower casing and removal of special characters)
  - Tokenization
  - Removal of stop words
  - Stemming / Lemmatization



Representing Text as Numbers

Northeastern University

- Common strategies for converting text into numerical vectors:
  - Bag of words model
  - TF-IDF vectors
  - Word embeddings

Bag of Words (BoW) Model

Northeastern University

- Each document is represented by a word counts vector
- Assign a fixed integer id to each word occurring in any document of the training set
  - e.g., by building a dictionary from words to integer indices
- For each document *i*, count the number of occurrences of each word *w* and store it in *X*[*i*, *j*] as the value of feature *j* where *j* is the index of word *w* in the dictionary
- *X* is called a **document-term matrix**
- This matrix is very sparse: most values of *X* are 0

	littl	hous	prairi	mari	lamb	silenc	twinkl	star
"Little House on the Prairie"	1	1	1	0	0	0	0	0
"Mary had a Little Lamb"	1	0	0	1	1	0	0	0
"The Silence of the Lambs"	0	0	0	0	1	1	0	0
"Twinkle Twinkle Little Star"	1	0	0	0	0	0	2	1

Document-Term Matrix

CountVectorizer

Northeastern University

class `sklearn.feature_extraction.text.` **`CountVectorizer`**(*, *input='content'*, *encoding='utf-8'*, *decode_error='strict'*, *strip_accents=None*, *lowercase=True*, *preprocessor=None*, *tokenizer=None*, *stop_words=None*, *token_pattern='(?u)\b\w\w+\b'*, *ngram_range=(1, 1)*, *analyzer='word'*, *max_df=1.0*, *min_df=1*, *max_features=None*, *vocabulary=None*, *binary=False*, *dtype=<class 'numpy.int64'>*) [source]

- Converts a collection of text documents to a matrix of token counts
- Also includes text preprocessing, tokenizing and filtering of stopwords

Argument	Description
lowercase	Convert all characters to lowercase before tokenizing
analyzer	Whether the feature should be made of word or character n-grams
stop_words	Can be ‘english’ or a list of stop words
token_pattern	Regular expression denoting what constitutes a “token”
ngram_range	The lower and upper boundary of the range of n-values for different n-grams to be extracted
max_df	Ignore terms that have a document frequency strictly higher than the given threshold
min_df	Ignore terms that have a document frequency strictly lower than the given threshold

CountVectorizer

Northeastern University

- Example using a sample corpus with 4 short documents:

```
corpus = [
    'This is the first document.',
    'This document is the second document.',
    'And this is the third one.',
    'Is this the first document?',
]
```

```
from sklearn.feature_extraction.text import CountVectorizer

count_vect = CountVectorizer()
X = count_vect.fit_transform(corpus)
```

```
X.shape
```

```
(4, 9)
```

CountVectorizer

Northeastern University

- Once fitted, the vectorizer has built a dictionary that maps words to feature indices:

```
count_vect.vocabulary_
```

```
{'this': 8,
 'is': 3,
 'the': 6,
 'first': 2,
 'document': 1,
 'second': 5,
 'and': 0,
 'third': 7,
 'one': 4}
```

- get_feature_names() returns array mapping from feature indices to words:

```
count_vect.get_feature_names()
```

```
['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']
```

CountVectorizer

Northeastern University

- CountVectorizer returns the document-term matrix as a **SciPy sparse matrix**
- Some Scikit estimator classes cannot handle sparse matrices (e.g., CategoricalNB)
- You can convert it to a standard NumPy array by calling its **toarray()** method

```
vocab = count_vect.get_feature_names()
pd.DataFrame(X.toarray(), columns=vocab)
```

	and	document	first	is	one	second	the	third	this
0	0	1	1	1	0	0	1	0	1
1	0	2	0	1	0	1	1	0	1
2	1	0	0	1	1	0	1	1	1
3	0	1	1	1	0	0	1	0	1

TF-IDF Model

Northeastern University

- Issues with the bag of words model:
  - Longer documents have higher average word count values than shorter documents, even though they might discuss the same topics
  - Some of the words appear in many documents in the corpus and are therefore less informative than those that occur in a small subset of the corpus
- TF-IDF stands for “Term Frequency times Inverse Document Frequency”
- TF-IDF scales down the impact of words that occur frequently in the corpus
- It combines two metrics:
  - tf – term frequency
  - idf – inverse document frequency

TF-IDF Model

Northeastern University

- Given a corpus of documents *D*
- For a term *t* in document *d* we compute:

$$\mathrm{tf}(t,d) = \frac{f_{t,d}}{|\{t' \in d\}|}$$

frequency of term t in document d

number of terms in document d

$$\mathrm{idf}(t) = \log \frac{|D|}{|\{d \in D | t \in d\}|}$$

number of documents that contain the term t

- Then tf-idf is calculated as:

$$\mathrm{tfidf}(t,d) = \mathrm{tf}(t,d) \cdot \mathrm{idf}(t)$$

Example of TF-IDF

Northeastern University

- Given the following term count tables of two documents:

Document 1

Term	Term Count
this	1
is	1
a	2
sample	1

Document 2

Term	Term Count
this	1
is	1
another	2
example	3

- The tf-idf of the word “example” is:

$$\text{tf}(\text{"example"}, d_1) = \frac{0}{5} = 0$$

$$\text{idf}(\text{"example"}) = \log\left(\frac{2}{1}\right) = 0.301$$

$$\text{tfidf}(\text{"example"}, d_1) = 0 \cdot 0.301 = 0$$

$$\text{tf}(\text{"example"}, d_2) = \frac{3}{7} \approx 0.429$$

$$\text{tfidf}(\text{"example"}, d_2) = 0.429 \cdot 0.301 = 0.129$$

TfidfVectorizer

Northeastern University

class `sklearn.feature_extraction.text.` **`TfidfVectorizer`**(*, *input='content'*, *encoding='utf-8'*, *decode_error='strict'*, *strip_accents=None*, *lowercase=True*, *preprocessor=None*, *tokenizer=None*, *analyzer='word'*, *stop_words=None*, *token_pattern='(?u)\b\w\w+\b'*, *ngram_range=(1, 1)*, *max_df=1.0*, *min_df=1*, *max_features=None*, *vocabulary=None*, *binary=False*, *dtype=<class 'numpy.float64'>*, *norm='l2'*, *use_idf=True*, *smooth_idf=True*, *sublinear_tf=False*) [source]

- Converts a collection of text documents to a matrix of TF-IDF features
- Equivalent to CountVectorizer followed by TfidfTransformer
- It has similar parameters to CountVectorizer

TfidfVectorizer

Northeastern University

- Example using the previous sample corpus:

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vect = TfidfVectorizer()
X = tfidf_vect.fit_transform(corpus)
```

```
vocab = tfidf_vect.get_feature_names()
pd.DataFrame(X.toarray(), columns=vocab)
```

	and	document	first	is	one	second	the	third	this
0	0.000000	0.469791	0.580286	0.384085	0.000000	0.000000	0.384085	0.000000	0.384085
1	0.000000	0.687624	0.000000	0.281089	0.000000	0.538648	0.281089	0.000000	0.281089
2	0.511849	0.000000	0.000000	0.267104	0.511849	0.000000	0.267104	0.511849	0.267104
3	0.000000	0.469791	0.580286	0.384085	0.000000	0.000000	0.384085	0.000000	0.384085

Example: Text Classification

Northeastern University

- Let's define a pipeline that combines TF-IDF vectorizer and a Naïve Bayes classifier
- The variant of NB most suitable for word counts / TF-IDF is multinomial Naïve Bayes

```
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB

clf = Pipeline([
    ('tfidf', TfidfVectorizer()),
    ('clf', MultinomialNB())
])
```

- We can now train the model with a single command:

```
clf.fit(twenty_train.data, twenty_train.target)
```

```
Pipeline(steps=[('tfidf', TfidfVectorizer()), ('clf', MultinomialNB())])
```

Example: Text Classification

Northeastern University

- Evaluation of the performance on the training and test sets:

```
clf.score(twenty_train.data, twenty_train.target)
```

```
0.9326498143892522
```

```
clf.score(twenty_test.data, twenty_test.target)
```

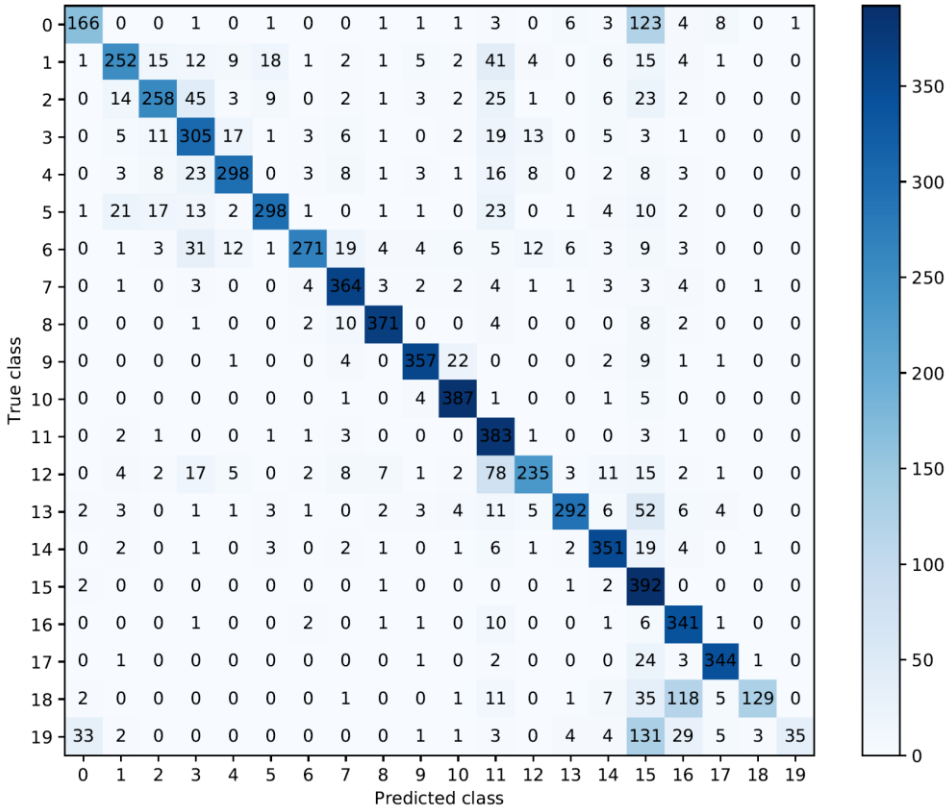
```
0.7738980350504514
```

Example: Text Classification

Northeastern University

- Confusion matrix between the true and predicted labels on the test set:

```
from sklearn.metrics import confusion_matrix

test_pred = clf.predict(twenty_test.data)
conf_mat = confusion_matrix(twenty_test.target, test_pred)
plt.figure(figsize=(10, 8))
plot_confusion_matrix(conf_mat)
plt.savefig('figures/nb_conf_matrix.pdf')
```



- Posts from the newsgroups on atheism (#0) and Christianity (#15) are often confused

Example: Text Classification

Northeastern University

- We can now use the model to predict the category of new documents:

```
new_docs = ['Fix the screen resolution', 'God is love', 'A car for sale']
new_pred = clf.predict(new_docs)

for doc, category in zip(new_docs, new_pred):
    print(doc, '=>', twenty_train.target_names[category])
```

```
Fix the screen resolution => comp.sys.mac.hardware
God is love => soc.religion.christian
A car for sale => misc.forsale
```

Handling Continuous Attributes

Northeastern University

- There are two ways to handle continuous attributes in Naïve Bayes models:
  - Discretize each continuous attribute
  - Assume the continuous attribute has a certain form of probability distribution and estimate the parameters of the distribution from the training set
    - Typically a normal distribution is assumed

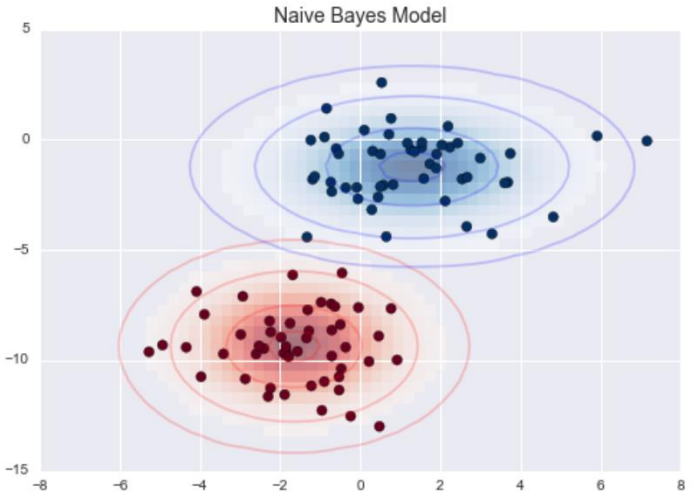
Gaussian Naïve Bayes

Northeastern University

- The distribution of the features given the class is assumed to be Gaussian:

$$p(x_j|y=k) = \frac{1}{\sqrt{2\pi}\sigma_{jk}} \exp\left(-\frac{(x_j-\mu_{jk})^2}{2\sigma_{jk}^2}\right)$$

- The MLE estimates of the parameters μ_{jk}, σ_{jk} are the mean and standard deviation of feature x_j over all the data points that belong to class y in the training set

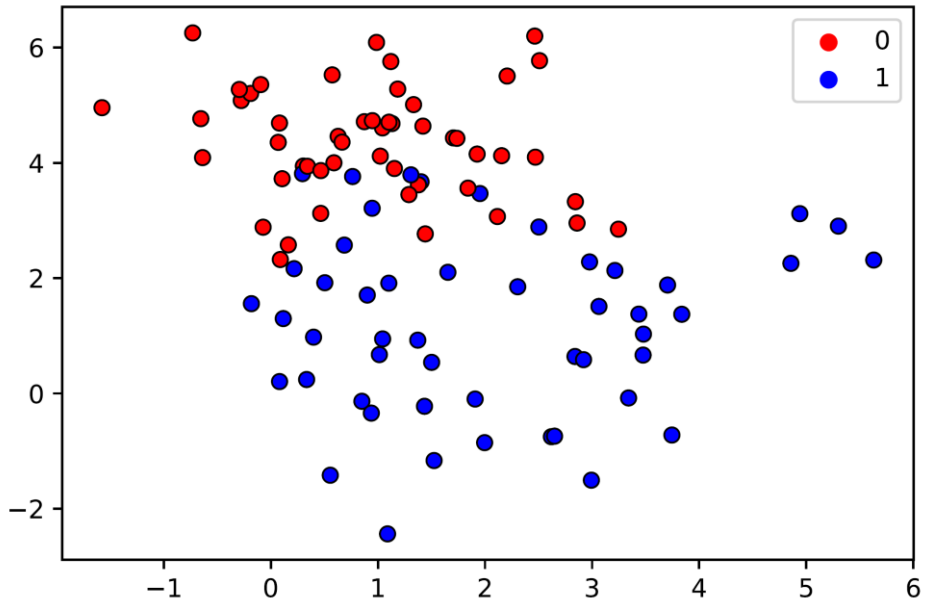


Gaussian Naïve Bayes

Northeastern University

- For example, imagine that you have the following data:

```
from sklearn.datasets import make_blobs

X, y = make_blobs(100, 2, centers=2, cluster_std=[1, 1.5], random_state=0)
sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=y, palette=['r', 'b'],
                edgecolor='black')
```



Gaussian Naïve Bayes

Northeastern University

- We first split the data into training and test sets:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=0)
```

- We now use GaussianNB to fit a Gaussian Naïve Bayes model to the data:

```
from sklearn.naive_bayes import GaussianNB

clf = GaussianNB()
clf.fit(X_train, y_train)
```

```
GaussianNB()
```

- The scores on the training and test sets are:

```
clf.score(X_train, y_train)
```

```
0.8875
```

```
clf.score(X_test, y_test)
```

```
0.9
```

Gaussian Naïve Bayes

Northeastern University

- We can now use the fitted model to predict the class label for new samples:

```
clf.predict([[1, 2]])
```

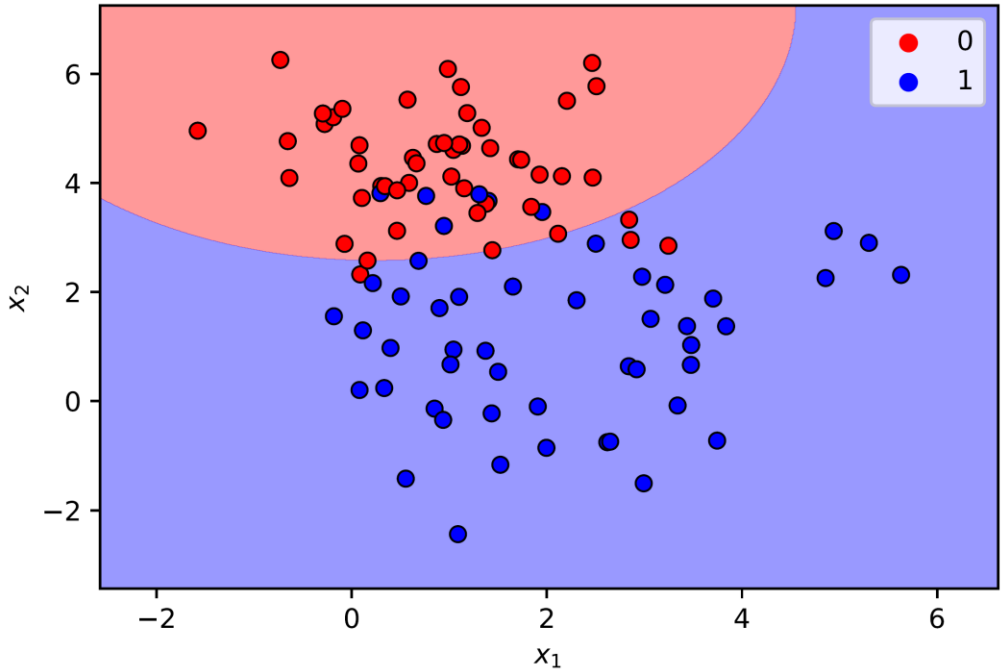
```
array([1])
```

```
clf.predict_proba([[1, 2]])
```

```
array([[0.17392267, 0.82607733]])
```

Gaussian Naïve Bayes

- We can also plot the decision boundary between the two classes:



- In general, the decision boundary will have a quadratic form
  - Unless the features have the same covariance matrix, in which case the quadratic part cancels out and the decision boundary is linear

Heterogonous Data Sets

Northeastern University

- The following data set contains both categorical and continuous attributes
- In this case we can compute the class-conditional probability for each categorical attribute, along with the sample mean and variance for the continuous attributes

Tid	Home Owner	Marital Status	Annual Income	Defaulted Borrower
1	Yes	Single	125K	**No**
2	No	Married	100K	**No**
3	No	Single	70K	**No**
4	Yes	Married	120K	**No**
5	No	Divorced	95K	**Yes**
6	No	Married	60K	**No**
7	Yes	Divorced	220K	**No**
8	No	Single	85K	**Yes**
9	No	Married	75K	**No**
10	No	Single	90K	**Yes**

P(Home Owner=Yes|No) = 3/7
P(Home Owner=No|No) = 4/7
P(Home Owner=Yes|Yes) = 0
P(Home Owner=No|Yes) = 1
P(Marital Status=Single|No) = 2/7
P(Marital Status=Divorced|No) = 1/7
P(Marital Status=Married|No) = 4/7
P(Marital Status=Single|Yes) = 2/3
P(Marital Status=Divorced|Yes) = 1/3
P(Marital Status=Married|Yes) = 0

For Annual Income:
If class=No: sample mean=110
sample variance=2975
If class=Yes: sample mean=90
sample variance=25

Heterogonous Data Sets

Northeastern University

- Predict the class label of a test sample
  x = (Home Owner = No, Marital Status = Married, Income = $120K)
- Using the information in the previous figure, the class-conditional probabilities are:

$$P(\mathbf{x}|\text{No}) = P(\text{Home Owner} = \text{No}|\text{No})P(\text{Status} = \text{Married}|\text{No})P(\text{Annual Income} = \$120\text{K}|\text{No})$$
$$= 4/7 \cdot 4/7 \cdot 0.0072 = 0.0024$$
$$P(\mathbf{x}|\text{Yes}) = P(\text{Home Owner} = \text{No}|\text{Yes})P(\text{Status} = \text{Married}|\text{Yes})P(\text{Annual Income} = \$120\text{K}|\text{Yes})$$
$$= 1 \cdot 0 \cdot 1.2 \cdot 10^{-9} = 0$$

- The prior probabilities of the classes are *P*(Yes) = 0.3 and *P*(No) = 0.7
- Therefore, the posterior probabilities are:

$$P(\text{No}|\mathbf{x}) = \alpha \cdot 0.7 \cdot 0.0024 = 0.0016\alpha$$
$$P(\text{Yes}|\mathbf{x}) = \alpha \cdot 0.3 \cdot 0 = 0$$

- Since *P*(No|**x**) > *P*(Yes|**x**), the sample is classified as No

Naïve Bayes Summary

Northeastern University

Pros

- Very fast both in training and prediction
- Provide class probability estimates
- Easily interpretable
- Robust to noise
  - Noise points are averaged out when estimating the conditional probabilities
- Can handle missing values
  - Missing values are ignored when computing the conditional probability estimates
- Robust to irrelevant attributes
- Works well with high-dimensional data
- Very few hyperparameters

Cons

- Relies on the Naïve Bayes assumption
  - Correlated attributes can degrade the performance of the classifier
- Cannot handle continuous attributes without assumptions on the distribution
- Generally doesn't perform as well as more complex models
- Can be used only for classification